LINEAR ALGEBRA Assignment 3 Report

110062219 翁君牧

December 13, 2022

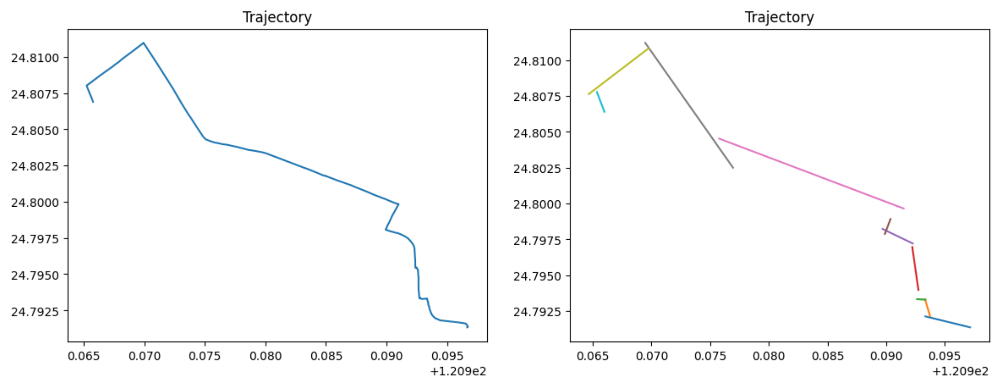
Contents

1 Break the Trajectory into Line Segments

The route I chose was from my dormitory to a ramen restaurant with 213 track point on the way. I broke the 11 turning points by hands, getting the following results shown in Figure 1.

2 Linear Least Square from `numpy`

In the previous assignment, we used `np.linalg.lstsq()` to solve overdetermined system. This time, we are to perform the linear regression by means of this function.



(a) Original

(b) Manually broken

Figure 1: Trajectory Results

2.1 How to know the errors?

By the documentation, `np.linalg.lstsq()` computes the approximate solution $\mathbf{x}$ to the system $A\mathbf{x} = \mathbf{b}$ and returns 4 things: the solution $\mathbf{x}$, the *"residuals"*, the rank and the singular values of A. The *"residuals"* are defined as "squared Euclidean 2-norm for each column in $\mathbf{b} - A\mathbf{x}$". Since we always has $\dim(\mathbf{b}) = 1$, the first and the only entry of the *"residuals"* is exactly what we want, $||\mathbf{b} - A\mathbf{x}||^2$.

In the notebook, I verified that the *"residuals"* equaled the error for a matrix A of size 1024×1000.

2.2 What's the meaning of `rcond`?

Just as I mentioned in the previous report, what `np.linalg.lstsq()` does is actually to compute the rank and "pseudo inverse" of a matrix via singular value decomposition.

So `rcond` is the threshold rate that if a singular value is less than the largest one, it would be treated as 0 during the process. Since the matrix is always overdetermined, it's pretty fine for us to opt out this feature.

For a matrix A of size 1000×1024, I measured the average time of 7 runs (10 loops each), `rcond` ranging from 10^{-8} to $2^{20} \times 10^{-8}$. Their time were so close that the difference could be hardly distinguished. Still, the result was plotted in Figure 2.

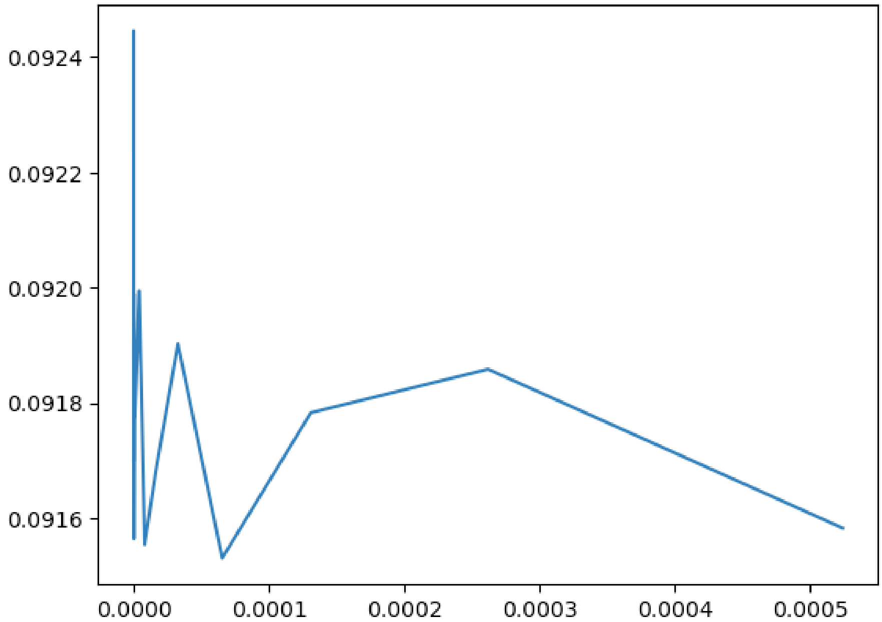


Figure 2: Average time w.r.t. `rcond`

3 Algorithm that Compresses GPS Data

My algorithm was described in the 11^{th} cell of the notebook. I maintain the earliest point p_i that is not done yet. For each point p_j, if the error of regression segment for $p_i, \dots, p_j$ is greater than ϵ, then p_j should be a turning point.

For the purpose of making all line segments connected, I add additional line segments between the ends.

3.1 Time Complexity

If there are m points, then A would be of size m by 2. Thus the time complexity for a `np.linalg.lstsq()` call would be $O(m \times 2^2) = O(m)$.

Generally, `np.linalg.lstsq()` would be called $O(n)$ times. In the worst case, there would be $O(n)$ points to do regression each time, thus the total time complexity would be $O(n^2)$. In general, if each segment contains at most m points, then the time complexity would be $O(nm)$.

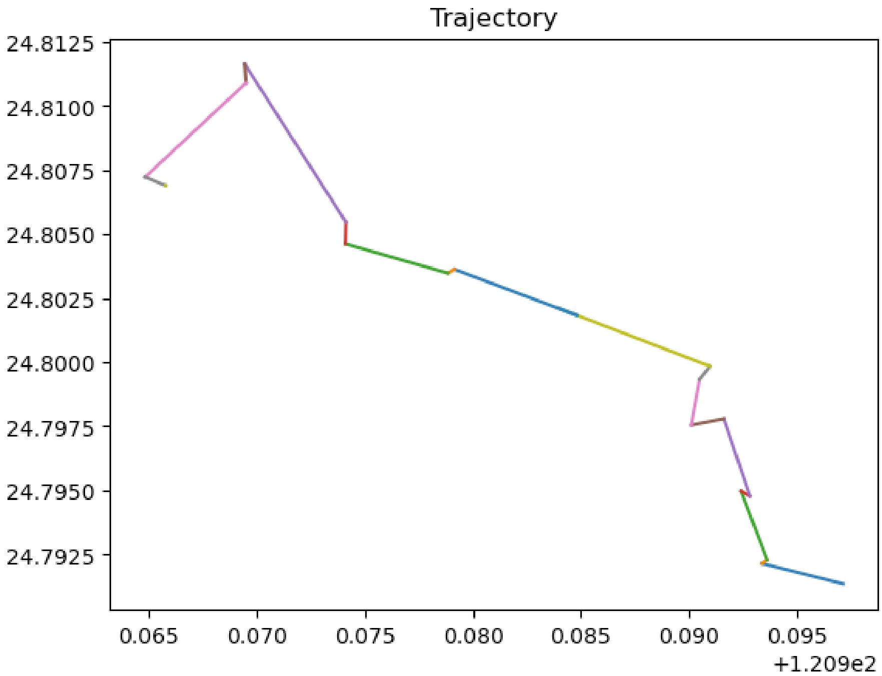


Figure 3: Trajectory broken by the algorithm, $\epsilon = 3.375 \times 10^{-6}$

3.2 Compression Efficiency

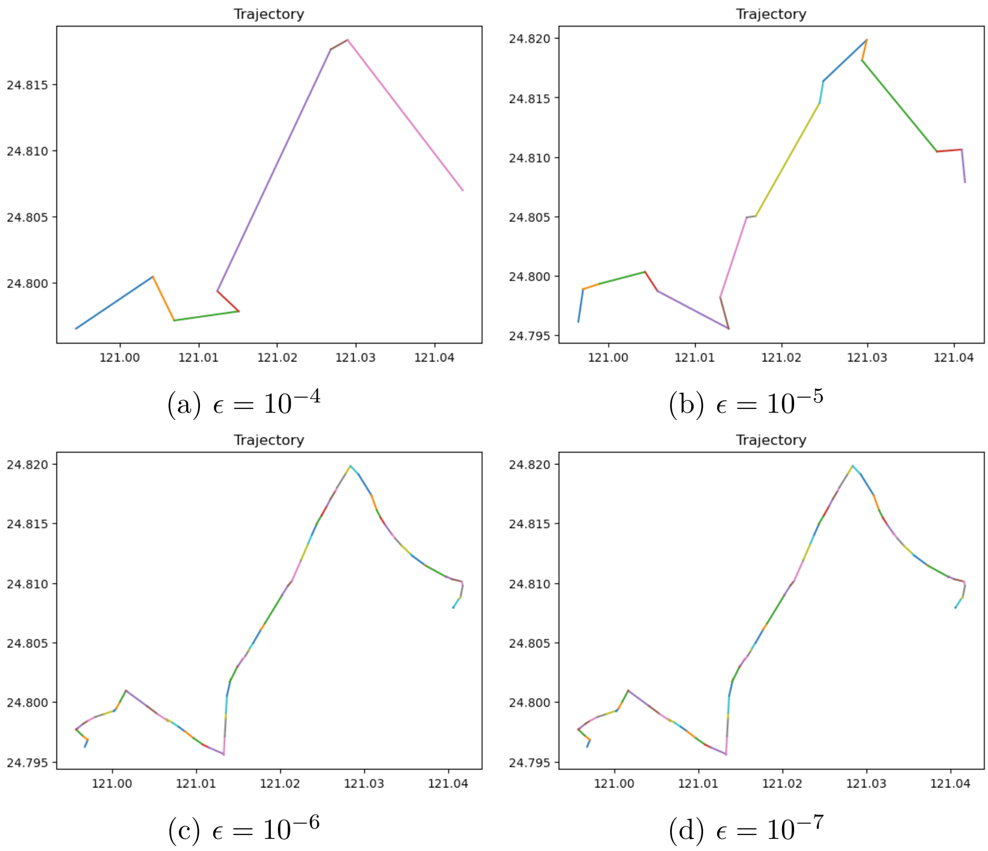
Let's discuss for the specific case, the route provided, the points required with regard to ϵ.

Table 1: Compression Efficiency

ϵ	# points
10^{-4}	8
10^{-5}	16
10^{-6}	39
10^{-7}	82

4 Parabolae Fittings

The matrix A provided indicates the descending order of coefficients for the linear function. Yet I prefer ascending order so that A would be a **Vandermonde matrix**.



(a) $\epsilon = 10^{-4}$ (b) $\epsilon = 10^{-5}$

(c) $\epsilon = 10^{-6}$ (d) $\epsilon = 10^{-7}$

Figure 4: Trajectory of given route broken by the algorithm

4.1 Derivations

So for the parametric equations of x, y, $\begin{cases} x = a_0 + a_1 t + a_2 t^2 \\ y = b_0 + b_1 t + b_2 t^2 \end{cases}$, we have the quadratic regression model in this linear system:

$$A'w' = X - \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}, A'v' = Y - \begin{pmatrix} \epsilon_1' \\ \epsilon_2' \\ \vdots \\ \epsilon_n' \end{pmatrix}$$

, where $A' = \begin{pmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ \vdots & \vdots & \vdots \\ 1 & t_n & t_n^2 \end{pmatrix}, w' = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix}, v' = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}$.

Our goal is to find solutions to w', v' that minimize the error $\sum_{k=1}^{n} \epsilon_k^2 + \sum_{k=1}^{n} {\epsilon_k'}^2 = ||A'w' - X||^2 + ||A'v' - Y||^2$, which is also the sum of square of their 2-dimension norms. Thus we could solve these by *linear least square* again.

4.2 Curves

I updated the algorithm to break the trajectory. Yet the function to draw the curves wasn't very well. Maybe I should connect the spaces between parabolae by some straight line segments or smooth curves.

The results are shown in Figure 5 & 6.

5 Total Least Square

There are various sort of *linear least squares*. The most common one, *ordinary least squares*, is aimed at finding a regressive line to predict the response variables. As a consequence, it only consider the distance in y-axis.

On the other hand, *total least squares*, a generalization of **orthogonal regression**, takes the errors for covariate variables into account. That is, our objective is to find a regressive line that the orthogonal distances between all points and the line is minimized.

`scipy` provides a module that dedicates to perform the **orthogonal distance regression**. There are inbuilt linear, quadratic, exponential models. Yet I tried linear one only.

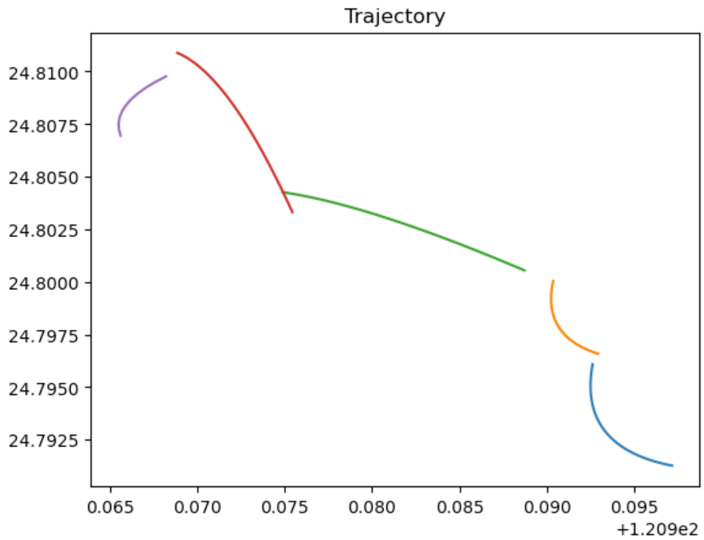
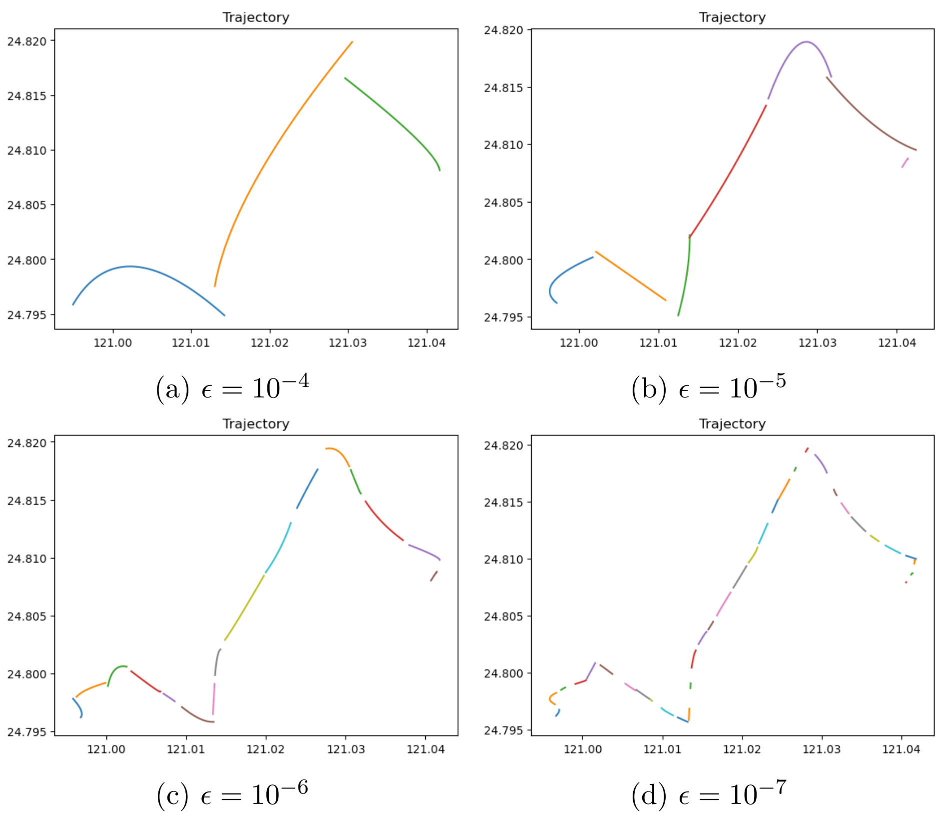


Figure 5: Parabola fitting of my route with $\epsilon = 10^{-5}$



(a) $\epsilon = 10^{-4}$

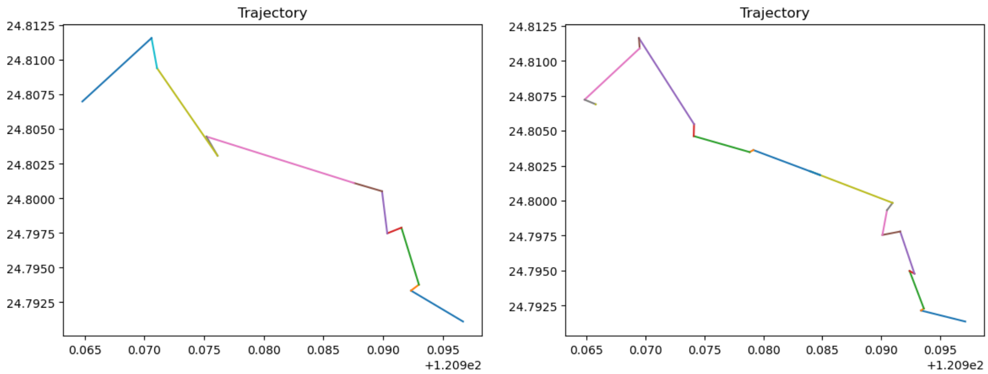
(b) $\epsilon = 10^{-5}$

(c) $\epsilon = 10^{-6}$

(d) $\epsilon = 10^{-7}$

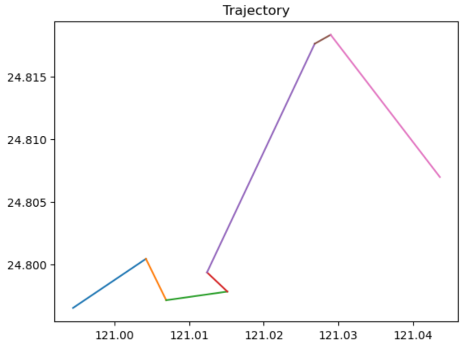
Figure 6: Parabolae fittings of given route with different ϵ

We could see that in Figure 8, if we take $\epsilon \leq 10^{-5}$, then the results of *OLS* and *ODR* are almost the same; when $\epsilon \geq 10^{-6}$, their difference are not distinguishable at the first glance.

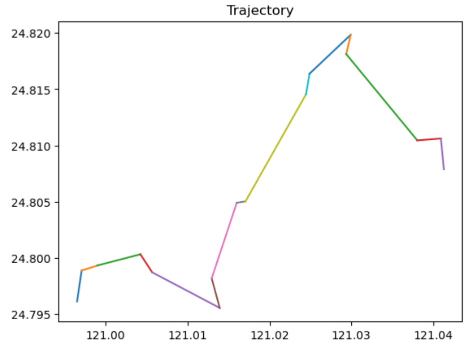


(a) $\epsilon = 10^{-6}$ (b) $\epsilon = 3.375 \times 10^{-6}$

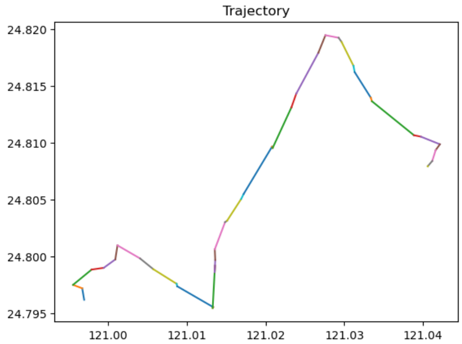
Figure 7: ODR of my route



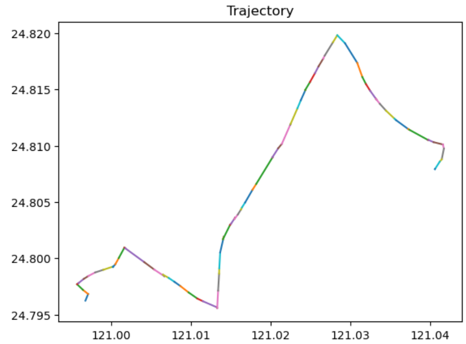
(a) $\epsilon = 10^{-4}$



(b) $\epsilon = 10^{-5}$



(c) $\epsilon = 10^{-6}$



(d) $\epsilon = 10^{-7}$

Figure 8: ODR of given route with different ϵ

Acknowledgements

I thank to National Center for High-performance Computing *(NCHC)* for providing computational and storage resources.